Similarities of Hydrodynamics and Chevron Kernels

Jeffrey D. Oldham

2001 Apr 20

1 Both Kernels Compute Flux Across Boundaries

Both the hydrodynamics kernel (based on [CBSW98]) and the Chevron kernel [LTD99] mainly consist of computing normals for and flux across cell boundaries. The algorithm steps consist of

1. Compute cell or subcell-centered values. For the Chevron kernel, use a set of linear equations to compute the pressure gradient ∇P for each subcell. For the hydrodynamics code, compute the cell-centered pressure.

2. Compute the cells' faces' outward normals. The normals are perpendicular to the cell faces and have lengths proportional to the faces' areas.

3. Compute corner fluxes f_v^c. Each *corner* corresponds to the space specified by the center c of a cell and one of its vertices v. Each corner flux is a measure of the amount of material leaving the cell's (outer) boundary through this corner. The cell's boundary intersects only half of the corner's faces. (The corner flux can either be a scalar or vector quantity so I do not know the proper nomenclature.)

   For the Chevron kernel, the corner flux f_v^c is

   $$(\mathbf{K_c} \cdot \nabla P_{c,v}) \cdot \vec{n}_{c,v}.$$

   $\mathbf{K_c}$ represents the permeability tensor for cell c. $\nabla P_{c,v}$ represents the constant pressure gradient in the c-v corner. $\vec{n}_{c,v}$ represents the sum of the normals of c-v corner's faces that intersect cell c's boundary.

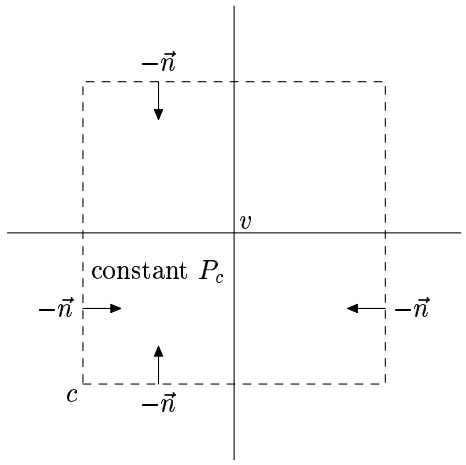

Figure 1: Discretized version of $\oint_{\partial V_v} P d\vec{S}$.

For the hydrodynamics kernel, the corner flux is

$$P_c \vec{n}_{c,v}.$$

P_c denotes the (constant) pressure in the cell c.

As written above, each corner flux is a measure of the amount of material leaving the cell's (outer) boundary through the corner. We present the derivation for Caramana et al. [CBSW98], Eq. (7)

$$M_v \frac{d\vec{v}_v}{dt} = -\int_{V_v} \vec{\nabla} P dV = -\oint_{\partial V_v} P d\vec{S} = \sum_c f_c^v \tag{7}$$

to illustrate why corner fluxes are important concepts. The first equality follows from integrating the time-evolution momentum equation throughout a cell V_v centered around v. Green's Theorem implies the second equality. ∂V_v indicates the boundary of V_v.

Discretization yields the last equality. Figure 1 illustrates how $\oint_{\partial V_v} P d\vec{S}$ is discretized. Each c,v-corner has a constant pressure P_c. Instead of integrating over the boundary of V_v, normals to each face are constructed with lengths equaling the face's area. The negative sign before the integral sign reverses the normals' directions.

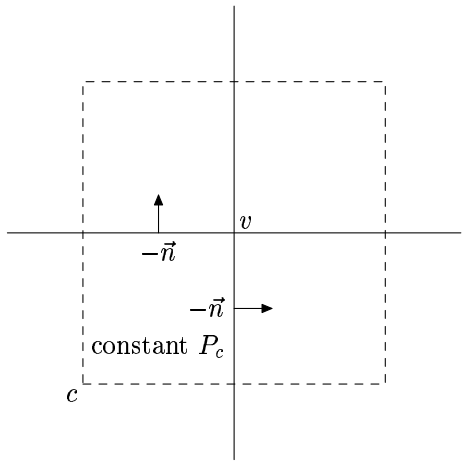

Figure 2: The previous figure redrawn to use c-centered coordinates.

To ease computation, a switch from v-centered coordinates to c-centered coordinates occurs. See Figure 2. Since parallelepipeds are used, the new normals parallel the normals in Figure 1 and have the same magnitude. Since the pressure P_c is assumed constant throughout the corner, the sum's value does not change.

Figure 3 indicates the result of multiplying the corner's pressure with the sum of the normals of each corner's faces. Thus, each corner flux approximates the amount of material leaving the cell's boundary through the corner.

4. The corner fluxes are summed. In the Chevron kernel, all corner fluxes in a cell c are summed. For the hydrodynamics algorithm, sums of all corners around vertices v and of all corners around a cell c are formed. In the latter sum, the dot products of the corner fluxes and v-centered velocities are the addends.

2 Implementing Flux Computations

2.1 Computing Corner Fluxes

Both algorithms compute corner fluxes, one for each intersection of cell-centered and vertex-centered cells. We will use a cell-centered field with twice the granu-

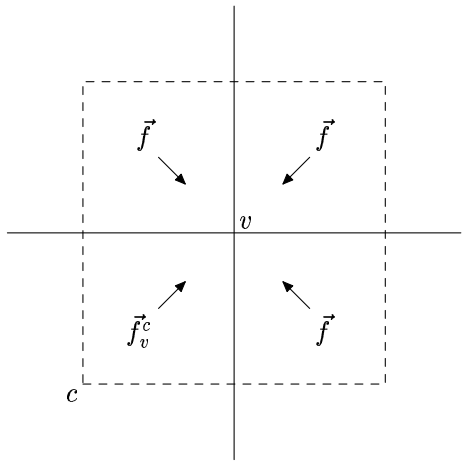

Figure 3: Adding each corner's normals yields corner fluxes.

larity in all dimensions to store these corner fluxes. We say this field has twice the granularity and is fine grained. When we refer to a cell in the coarser field without specifying its centering, assume it is a cell-centered cell.

The second algorithmic step is to compute the outward normals of each cell's faces. A naïve implementation would independently compute each cell's normals. By construction, the faces of two adjacent cells are the same but have opposite normals so we can reduce the computation by having each cell compute only its normals for positive-dimension faces. These would be stored in a face-centered field. A *face* is a $d-1$-D polytope that is a subset of a d-D polytope. Each face can be specified by a dimension and a positive or negative indication. (Scott Haney proposed computing the area, not a normal, for each positive-dimension face. Although sufficient for the Chevron kernel where cell vertices do not move, vertices move in the hydrodynamics code. Thus, normals, not just areas, need to be computed.)

Instead of directly computing corner fluxes, we recommend computing corner normals. A *corner normal* is the sum of the normals of a corner's faces that intersect the cell's boundaries. Starting with normals in a face-centered coarse field, we use an interpolation operator to compute a cell-centered finer field. For each cell in the finer field, the sum of the incident normals is computed:

```
cornerNormals = addNormals<2,Cell,AllFace>(normals);
```

The first two template parameters of `addNormals` indicate that the resulting

field has twice the granularity in all dimensions and is cell-centered. The third parameter says that the `normals` field is face-centered.

For the Chevron kernel, the corner flux f_v^c is $(\mathbf{K_c} \cdot \nabla P_{c,v}) \cdot \vec{n}_{c,v}$. The parenthesized term can be stored in a cell-centered fine-grained field. To compute the term, use

```
cornerFlux = dot(dot(replicate<2>(permeability), pressure-
Gradient),
  addNormals<2,Cell,AllFace>(normals));
```

`replicate<2>(permeability)` forms a field with twice the granularity where values are duplicated in each dimension and the centering is the same. The other operations are explained above or self-explanatory.

For the hydrodynamics kernel, the corner flux $P_c \vec{n}_{c,v}$ can be computed as

```
cornerForce = replicate<2>(pressure) *
  addNormals<2,Cell,AllFace>(normals);
```

`replicate<2>(pressure)` forms a cell-centered field with twice the granularity where values are duplicated in each dimension. The second operand is explained above.

2.2 Summing Corner Fluxes

For the Chevron kernel, the sum of all corner fluxes in a cell c are stored in a cell-centered, coarse-grain field.

```
fluxes = total<2>(cornerFlux);
```

The restriction operator `total<2>` sums all cells in a fine-grained field corresponding to one cell in a coarse-grained field, where the fine-grained field has twice the granularity in every dimension. Since no centerings are specified, the two fields have the same centerings.

For the hydrodynamics kernel, sums of all corner fluxes around vertices v are formed.

$$\Delta \vec{v}_p = k_1 \sum_z \vec{f}_z^{p,\sigma} \tag{1}$$

Code for this sum is

```
velocityChange = constant1 * total<2,Vert,Cell>(forces);
```

As for `addNormals` above, the three parameters indicate that the input field `forces` is cell-centered and has twice the granularity. The second parameter ensures the resulting field is vertex-centered. Since there are more vertex-centered values than cell-centered values, one layer of `force`'s guard cells is used in the computation.

Sums of dot products of corner fluxes and velocities are also computed.

$$\Delta e_z = -k_2 \sum_p \vec{f}_p^z \cdot \vec{v}_p^{n+1/2} \tag{2}$$

Corresponding code is

```
internalEnergy += constant2 * total<2>(dot(cornerForce,
  replicate<2,Cell,Vert>(velocity + velocityChange)));
```

The `cornerForce` field is cell-centered so the `replicate`d field must also be cell-centered. The velocity sum field is vertex-centered, as indicated by the third template parameter. For each value in a coarse vertex-centered field, the vertices in the corresponding polytope in a vertex-centered replicated field all have the same value. This polytope's "smallest" vertex matches the vertex in the coarse field. To form a cell-centered field, shift all values half a unit in negative directions.

A Centerings

When thinking about converting between centerings, we needed to know possible centerings for d-D polytopes and how the number of centers as a function of the dimension d. We present a consistent notation for and counting of centerings of polytopes of various dimensions.

A *d-D polytope* is recursively defined. A *0-D polytope* is a point with a vertex label of $\{\}$. To form a d-D polytope, start with a $(d-1)$-D polytope and make a copy of it. In the original, append 0 to all vertex labels. In the copy, append 1 to all vertex labels. Connect vertices with labels differing only in the last digit to form the d-D polytope.

The number of vertices in a d-D polytope is 2^d, as demonstrated by the construction. By construction, vertices with vertex labels differing in one bit are connected by an edge.

Intuitively, centerings are symmetrically placed points in a polytope. For a 3-D polytope, the vertex-centering consists of all vertex singleton sets $\{\{000\}, \{001\}, \{010\}, \{011\}, \{100\}, \{1$ An edge-centering has one point at the center of each edge. Each edge is specified by two adjacent vertices. These twelve centerings include $\{000, 001\}$, $\{010, 011\}$, $\{000, 010\}$, $\{001, 011\}$. There are six face-centerings, each specified by four vertices. For example, $\{000, 010, 100, 110\}$ and $\{001, 011, 101, 111\}$ are faces. The one cell-centering is specified by all eight vertices: $\{000, 001, 010, 011, 100, 101, 110, 111\}$.

To specify centerings for a d-D polytope, we will use strings of 'c' and 'v' characters and having d length. For example, in a 3-D polytope, the centerings are

vvv	vertex-centered
vvc, vcv, cvv	edge-centered
vcc, cvc, ccv	face-centered
ccc	cell-centered.

The strings are recursively constructed. An empty string corresponds to an empty set. Appending a 'v' doubles the number of sets, appending 0 to the vertex labels of all elements in the first sets and 1 to all elements in the second sets. Thus the number of connections 'c' between sets is not increased. Appending a 'c' duplicates the elements within a set by appending 0 to half of the elements and 1 to the copies. Thus, the number of vertices connected together to form the unit, e.g., edge or face, is doubled but the number of sets remains the same.

For example, "vvc" is the set $\{\{000, 001\}, \{010, 011\}, \{100, 101\}, \{110, 111\}\}$. Since the one 'c' is in the z-dimension position, the set consists of edges traversing the z-direction, i.e., vertical edges. The two "vcc" faces $\{000, 001, 010, 011\}$ and $\{100, 101, 110, 111\}$ connect vertices in y- and z-directions so they are the "x-faces" of a cube.

All strings with the same numbers of 'c's and 'v's are commonly grouped into a centering. That is, ordering of 'c's and 'v's is ignored.

Using these strings, we can count the number of centerings and the number of points within a centering. Since there are $d + 1$ ways to form order-independent length-d strings of 'c's and 'v's, there are $d+1$ centerings for a d-D polytope. "vv ... v" is called a vertex-centering. "cc ... c" is called a cell-centering. Strings of 'v's with exactly one 'c' specify edge centerings. Strings of 'c's with exactly one 'v' specify face centerings. That is, a face is a $d - 1$ polytope.

The number of units in a centering is $\binom{d}{|c|}2^{|v|}1^{|c|}$, where $|v|$ and $|c|$ indicate the number of 'v's and 'c's in the string. The number of vertices in such a unit is $1^{|v|}2^{|c|}$. For example, the number of faces is $2d$, while the number of vertices is 2^d. The faces *incident* to a particular vertex are those faces that contain that vertex.

B Acknowledgements

Thanks to Scott Haney for explaining the centering issue and posing the right questions as well as his clear presentation of the Chevron algorithm. Thanks to John Hall for his insight on centerings.

References

[CBSW98] E. J. Caramana, D. E. Burton, M. J. Shashkov, and P. P. Whalen. The construction of compatible hydrodynamics algorithms utilizing conservation of total energy. *Journal of Computational Physics*, 146:227–262, 1998.

[LTD99] S. H. Lee, H. Tchelepi, and L. J. DeChant. Implementation of a flux-continuous finite difference method for stratigraphic, hexahedron grids. In *1999 SPE Reservoir Simulation Symposium*, number SPE 51901, 08–11 February 1999.